INVENTORS: John R. Hind, Thomas Schaeck, Brad B. Topol

Low-Latency, Incremental Rendering in a Content Framework

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to computer software, and deals more particularly with improved techniques for rendering content in a content framework (such as a portal page provided by a portal system).

Description of the Related Art

The popularity of distributed computing networks and network computing has increased tremendously in recent years, due in large part to growing business and consumer use of the public Internet and the subset thereof known as the "World Wide Web" (or simply "Web").

Other types of distributed computing networks, such as corporate intranets and extranets, are also increasingly popular. As solutions providers focus on delivering improved Web-based computing, many of the solutions which are developed are adaptable to other distributed computing environments. Thus, references herein to the Internet and Web are for purposes of illustration and not of limitation.

The early Internet served primarily as a distributed file system in which users could request delivery of already-generated static documents. In recent years, the trend has been to add more and more dynamic and personalized aspects into the content that is served to requesters. One area where this trend is evident is in the increasing popularity of content frameworks such as those commonly referred to as "portals" (or, equivalently, portal systems or portal servers). A portal is a type of content framework that serves as a gateway, or focal point, for users to access an aggregation or collection of content from multiple sources. A portal provides its users with a Web page known as a "portal page", often structured as a single overview-style page (which may provide links for the user to navigate to more detailed information). Alternatively, portal pages may be designed using a notebook paradigm whereby multiple pages are available to the user upon selecting a tab for that page. Some experts predict that portal pages will become the computing "desktop" view of the future.

Portal pages offer users Web pages that contain content from many different sources, and provide rich content to users in a compact form. Sources of portal page content include Internet sites, a company's intranet, news groups, applications, and other content management feeds.

Many portals allow users to design a personalized version of the portal page, whereby the user can tailor the content, the layout, and/or the presentation attributes (such as color, font, etc.) of the page to his or her own preferences.

Portals are commonly designed using a component model that allows plugging components referred to as "portlets" (or, alternatively, components using a similar abstraction) into the portal infrastructure. Each portlet is responsible for obtaining a portion of the content that is to be rendered as part of the complete portal page for the user. The end result of the portal's content aggregation process is a Web page whose content is well suited for the needs of the portal user. Fig. 1 provides an example of a portal page 100 which includes three portlets 110, 120, 130. Portlet 110 in this example displays news headlines. Portlet 120 shows a stock ticker for the user's favorite stocks, and portlet 130 displays the current weather and weather forecast for the user's selected city.

While portal pages, by their nature, are rich in content, they are not without their disadvantages. Obtaining the content for the rendering can be a time-consuming process. In order to create a page of aggregated content, the portal must execute each portlet, wait for it to obtain and tailor its content, and then splice this content together into a markup language document representing the portal page it intends to send (as a stream) to the requesting browser. As a result of this process, the portal page renderer of the portal is unable to send content to the browser until the content generated by each of the portlets has been obtained. Thus, if one portlet takes an unusually long amount of time to acquire its content (for example, due to Internet delays, a server being down, and so forth), the portal user experiences having to wait a long time to see any content at all from his or her portal page. The user may think the portal is broken, or may simply become frustrated with using the portal. Many portal sites command a great deal of advertising revenue, due to their popularity with large numbers of users. Enterprises providing portals must therefore strive to keep their users happy, including avoiding these types of delays.

One prior art approach to reducing the time a user waits for receiving a portal page is to spawn individual threads for each portlet. This introduces concurrency into the computing time on the portal, and helps to reduce latency to a certain extent. However, it is still the case that the portal page cannot be delivered to the browser for rendering to the user until all the portlets have acquired their content.

SUMMARY OF THE INVENTION

An object of the present invention is to provide improved techniques for rendering content in a content framework (such as a portal page provided by a portal system).

Another object of the present invention is to provide techniques for incrementally rendering portal pages to users.

A further object of the present invention is to provide end users with an improved experience when using portal pages.

Yet another object of the present invention is to enable delivering partially-complete portal pages to users, thereby reducing the user's wait time, and programmatically generating a mechanism for retrieving the remaining content.

It is another object of the present invention to provide improved portal page rendering with minimal disruption to existing end-user systems.

Other objects and advantages of the present invention will be set forth in part in the description and in the drawings which follow and, in part, will be obvious from the description or may be learned by practice of the invention.

To achieve the foregoing objects, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, systems, and computer program products for incrementally rendering content in a content framework. In one aspect, this technique comprises: receiving a request for a portal page, wherein one or more portlets provide content for the portal page; immediately returning a response message containing a first document, the first document representing results from portlets which have acquired their content; and programmatically generating a mechanism for delivering an updated document if the first document does not represent results of all portlets.

In another aspect, this technique comprises: receiving a request for a portal page, wherein one or more portlets provide content for the portal page; immediately returning a response message containing a first document, the first document representing results from portlets which have acquired their content; and automatically delivering an updated document if the first document does not represent results of all portlets.

In a further aspect, this technique comprises: receiving a request for a portal page frame, wherein one or more portlets provide content for the portal page frame; immediately returning a response message containing a first mini-document, the first document representing results from portlets which have acquired their content; and programmatically generating a mechanism for delivering an updated mini-document if the first mini-document does not represent results of all portlets.

The present invention may also be used advantageously in methods of doing business, for example by providing improved portal systems and/or services wherein the delivery of content from such systems/services occurs in an improved manner. Providers of such systems and services may offer their content-provider clients some assurances of improved initial content delivery time, such that the content providers can expect improved end-user viewing experiences.

The present invention will now be described with reference to the following drawings, in which like reference numbers denote the same element throughout.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates a portal page which aggregates content from three portlets, according to the prior art;

Figures 2A - 2C depict several representative examples of syntax that may be used to enable incremental rendering of portal pages, according to preferred embodiments of the present invention;

Figure 3A illustrates a sample portal page in which content from some portlets is rendered, while content from other late-finishing portlets is not, according to the present invention;

Figure 3B shows the sample portal page from Figure 3A after a second incremental delivery of portlet content has occurred, according to the present invention; and

Figures 4 - 6 provide flowcharts illustrating operation of preferred embodiments of the present invention.

DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention provides improved techniques for rendering content in a content framework. For purposes of illustration but not of limitation, the content framework is described herein as a portal, and the components that plug into this framework are described as portlets. Several alternative techniques are disclosed for incrementally delivering portlet output to a user's browser (or other user agent, equivalently), thereby reducing the time the user spends waiting for his or her portal page to be rendered. Preferred embodiments take advantage of existing browser capabilities, thereby enabling the advantages of the present invention to be realized without requiring modifications to end-user software or end-user devices. (In cases where an end user's browser does not support one or more of these capabilities, the advantages of the present invention may be realized after upgrading the browser or by modifying the browser to support the capabilities. Because the capabilities leveraged by the present invention are well known in the art, revisions to browser code to support the capabilities should be straightforward.)

The techniques of the present invention are directed toward providing an end user with a portal page immediately, where this initially-provided portal page contains content from those portlets for which content was available upon receiving the portal page request. This initial portal page is then updated to include the delayed portlet content at a later time. The update mechanism is programmatically generated by the portal, using one of several techniques, as will now be described.

In a first embodiment, a refresh header is used to inform the browser receiving a partially-complete portal page that a revised, or refreshed, version of the page should be requested after waiting for a specified length of time. Refresh headers are well known in the art, and are commonly supported by browsers as an extension of the Hypertext Transfer Protocol ("HTTP"). Use of refresh headers is sometimes referred to as a "client pull" technique, because the client (i.e. the browser) is responsible for requesting the revised content. In a second embodiment, multipart MIME ("Multi-purpose Internet Mail Extensions") messages are used to enable the portal to deliver revised content directly to the browser on the portal's initiative (e.g. as each portlet completes, or after all portlets complete, or perhaps after expiration of some time interval). In particular, the MIME type "multipart/x-mixed-replace" (or an analogous content type) is preferably used for this second embodiment. Use of this MIME type is sometimes referred to as a "server push" technique, because the server (i.e. the portal) pushes content to the browser under the server's control. Client pull and server push for delivering revised Web page content to a browser are described in "An Exploration of Dynamic Documents", published by Netscape Communications Corp. on the Internet at http://home.netscape.com/assist/net_sites/pushpull.html. The present invention defines novel approaches for adapting these techniques to use with portlets and portal pages.

Figs. 2A and 2B provide examples of syntax that may be used for incremental portal page rendering according to the first embodiment of the present invention. Fig. 2A illustrates an HTTP response message sent to a browser when transmitting an initial partially-complete portal page. In this example, the response header shown in Fig. 2A indicates the following information: (1) the status is "OK" (see element 205); (2) the content type of this response message is "text/html" (see element 210); and (3) the content delivered with this response message should be refreshed, for this example, after waiting 12 seconds (see element 215). An alternative syntax is shown in Fig. 2B, which uses a Hypertext Markup Language ("HTML") META tag with an HTTP-EQUIV attribute, as shown at 220. In general, a META tag may be used to identify properties of a document, and an HTTP-EQUIV attribute on a META tag may be used in markup language documents to explicitly specify information that an HTTP server should convey in the HTTP response message with which the document is transmitted. In the example of Fig. 2B, the value of the HTTP-EQUIV attribute is "Refresh", and the value of the CONTENT attribute is "12". Thus, this example syntax specifies that the content of the HTML document which includes the META tag should be refreshed after waiting for 12 seconds, and has an effect equivalent to the refresh header of Fig. 2A for browsers which support META tags in HTML documents. (META tags are supported by browsers implementing HTML 3.0 and above.) Information on the META tag and HTTP-EQUIV attribute can be found in Request For Comments ("RFC") 2518 from the Internet Engineering Task Force, which is entitled "HTTP Extensions for Distributed Authoring -- WEBDAV" (Feb. 1999), as well as on the Internet at location http://www.wdvl.com/Authoring/HTML/Head/Meta/HTTP.html.

The syntax forms shown in Figs. 2A and 2B are referred to equivalently hereinafter as "refresh headers". When a browser supporting refresh headers receives a document which has a refresh header, the browser automatically sends a subsequent request for refreshed content after waiting for the specified time (in seconds). The server receiving the subsequent request then returns the content, and that content is used to repaint the screen (in a visual rendering), overlaying the previous content.

Referring now to Fig. 3A, a sample portal page 300 is shown in which content from three portlets is to be rendered (see elements 310, 320, 330). Portlet 310 represents the user's calendar entries, portlet 330 provides a yearly scheduling chart, and portlet 320 is to provide up-to-date analysis of customer call frequency at a Help Desk. Suppose that analyzing the call frequency information takes a relatively long amount of time, and that the calendar and yearly chart have already been generated. Rather than delaying delivery of the entire portal page until all the portlet content is ready, as in the prior art, the partially-complete portal page 300 is delivered to the end user without waiting for portlet content at 320, which in this example represents the late-finishing portlet. If this page 300 is delivered with the refresh header of Fig. 2A or 2B, the user's browser will display this initial portal page, wait for 12 seconds, and then send another request for the same page. Assuming that the late-finishing portlet has acquired its content in the interim, the response to the second request will enable the browser to repaint the portal page with complete information. An example of the resulting refreshed portal page 340 is shown in Fig. 3B, where the customer call frequency analysis is presented as chart 350. This incremental rendering approach gives the end user at least some of the portal page content in a very quick response, providing a much more user-friendly experience.

Note that the placeholder message displayed instead of portlet content at 320 is merely one way in which absence of a portlet's content can be indicated. Alternatives include displaying an image to fill the space; displaying a message reflecting the portlet's purpose (which may be obtained programmatically from the portlet's properties interface, for example); and leaving the space unoccupied. (When the portlet content eventually arrives and is painted in its place on the portal page, the user ideally will perceive that the portlet's area of the portal page is simply being repainted with its content, rather than the entire page being repainted.)

It may happen that the delay value on the refresh header was not set to a sufficiently long value, and the request for portal page content that was triggered by the refresh header arrives at the portal server before all portlets have finished acquiring their content. When this happens, the server preferably sends a revised portal page (the second version of the page, that is) in response to the browser request, and includes another refresh header with this response. The time value on this header may be the same as, or different from, the value on the first refresh header. (The manner in which the time value for the header is determined is described below with reference to Fig. 4.) This process repeats until all content is ready; when the portal has a complete portal page for returning to the browser, the refresh header is omitted.

Turning now to Fig. 2C, an example is provided of syntax that may be used for incremental portal page rendering according to the second embodiment of the present invention. The MIME type of "multipart/x-mixed-replace" is used for the content type header in the HTTP response with which the portal page is delivered to the requester, as shown at 230, to indicate that content appearing within data blocks denoted by successive "boundary" elements is to replace the content rendered from earlier data blocks. The boundary attribute in this example specifies that the string "PortalPageContent" (see 240) is used to delimit the data blocks. When using this embodiment, an initial partially-complete portal page is specified as the first data block, and follows the first occurrence of the boundary delimiter. This portion 250 of the document shown in Fig. 2C is then transmitted to the browser, where it will be rendered. When additional portlet content is acquired, the markup language document specifying the portal page is regenerated to include the newly-acquired content. The boundary string, followed by the regenerated portal page, is then sent to the browser as a replacement document (illustrated by element 260 of Fig. 2C). This replacement document will be rendered, overlaying the previous version of the portal page. Suppose for purposes of the example that the second version of the portal page still did not include content from all the portlets, and that a third version is generated at some later time to include all the portlet content. This replacement content is again preceded by the boundary string, and because it is the final version of the portal page, it is also followed by the boundary string where the boundary string ends with two dashes ("--") to denote that this is the terminating boundary string (see element 270 in Fig. 2C).

In this second embodiment, the portal server is in control of when additional content is pushed to the client. Therefore, the server may choose to wait until all portlets have finished before sending a revised page, such that there are at most two data blocks in the multipart MIME document. Or, the server may send a revised data block each time it detects that a portlet has acquired its content. Other triggering events may also be used to send a revised data block, such as expiration of a timer after which a document including the currently-available content will be sent. (The value of this timer may be determined using the techniques described herein for setting the value on the refresh header. Refer to the discussion of Block 450 of Fig. 4, below, for more information.)

The portal page content delivered in subsequent requests, after receiving content specifying a refresh header according to the first embodiment or after receiving subsequent data blocks in the multipart MIME approach of the second embodiment, will overlay previously-rendered content from earlier versions of the portal page, as has been stated. Therefore, portal pages which are rendered using the approach of the first and second embodiments are preferably designed to limit the user's interaction with the page contents. Interactions may include clicking on links and so forth, but preferably do not allow the user to fill in forms or perform other types of interactions that would result in lost work when the page is overlaid. Or, rather than limiting the user's interactions in this manner, a third embodiment of the present invention may be used wherein portal pages are rendered using frames. This approach associates a separate Uniform Resource Locator ("URL") with each frame of the page, where the frame markups sent to the browser instruct the browser to retrieve a "mini-page" from that URL to fill in the frames in the page. In a first aspect of this third embodiment, each mini-page represents the results of a single portlet. In an optional second aspect of this third embodiment, a mini-page may represent an aggregation of portlets. When using the approach of the first aspect, the browser sends a separate request for the content of each portlet which has its own frame, and the request is automatically held at the server until the portlet's content is ready. At that time, the content is formatted and returned on a response message, and the receiving browser renders it in place within the already-rendered portal page. Thus, the rendered portal page is not overlaid when using this technique, except for the portlet's own frame.

There is a practical limit to the number of outstanding browser requests which a browser implementation can manage. If a portal page has a large number of portlets, this limit can be avoided using the technique of the second aspect, whereby a frame holds several portlets. In this case, when the browser sends a request for a URL representing more than one portlet, the server returns the content for as many portlets as are ready when the request is received. It then uses the refresh header technique of the first embodiment or the multipart MIME technique of the second embodiment to subsequently deliver a new version of the frame which includes the content from late-finishing portlets.

In a fourth embodiment of the present invention, if the content of all portlets is not ready when the immediate response is being formatted for delivery to the requesting browser, a hyperlink is programmatically inserted into the initial version of the markup language document which represents the portal page. Preferably, a descriptive message is also inserted, instructing the user that clicking on this hyperlink will request a revised version of the portal page (or when frames are used for the page, a revised version of the frame with which the hyperlink is associated). The URL for the hyperlink is then the same URL used to originally request the content of the portal page (or of the frame, respectively). Conversely, if a content request is received and all the content is ready (whether on the initial request or a subsequent request), the hyperlink is omitted.

Referring now to the flowcharts in Figs. 4 - 6, logic depicting operations of preferred embodiments of the present invention will now be described. Fig. 4 illustrates logic operating on a portal server when using the client pull approach of the first embodiment. At Block 400, a content request for a portal page is received from a user agent. The portlets comprising that portal page are then determined (Block 410). When users are allowed to personalize their portal pages to include different content, then the portal identification in Block 410 may comprise obtaining information identifying this particular user from the content request, and using that information to access a user preferences repository to locate this user's previously-stored preferences, including the set of portlets to be rendered for this user.

In Block 420, a test is made to determine if all portlets have their content available. If so, then processing continues at Block 430, described below. Otherwise, an identification is made of those portlets which have content available and those which need to fetch content. In preferred embodiments, portlet instances can be in one of three states: not ready, loading, or ready. Threads are spawned only for portlets in the "not ready" state (which is the initial state). When a thread is spawned, the portlet transitions from "not ready" to "loading". When a portlet has its content, it transitions from "loading" to "ready". Thus, at Block 440, separate threads are spawned for each portlet which requires remote content and which is in the "not ready" state, to maximize the concurrency of portlet operations in order to reduce the cumulative processing time and the corresponding delay in presenting a complete portal page to the user. (Some portlets may have locally-cached content, which may be indicated, for example, in the portlet deployment descriptors. In those cases, the content is preferably retrieved from cache without spawning a separate thread.)

Block 450 computes the refresh interval to be used for this response. In preferred embodiments, prior art techniques are preferably leveraged to develop heuristics for empirically determining how long it takes a portlet to fetch its content. (This is analogous to algorithms used by transmission protocols to determine round trip times.) In the preferred approach, timing measurements are taken for how long it takes a particular portlet to fetch its content. This value is referred to as the measured fetch time or "FT". The measured fetch time for a particular (n-th) invocation of a portlet is denoted as FT(n). An algorithm computes the predicted fetch time, "PT(i)", for the current (i-th) fetch operation. In preferred embodiments, this algorithm uses the previously-recorded measured fetch time for the fetch operation (i-1) and the fetch time that was predicted for that fetch operation. A weighting value is preferably applied to dynamically adjust the predicted fetch time for this i-th invocation of the portlet to take into account changing factors such as network congestion and server loads. Using a weighting value "a", where a is some number between 0 and 1, the algorithm used in preferred embodiments is:

$$PT(i) = a * FT(i-1) + (1-a) * PT(i-1)$$

A timer is preferably started each time a thread is spawned for a portlet. (When the thread completes and returns the portlet's content to the spawning process, referred to herein as the "portlet invoker", the finish time will be noted and compared to the start time to determine the current measured fetch time. The newly-computed measured fetch time will be stored, along with the predicted fetch time, to enable use of the algorithm shown above.)

The processing in Block 450 comprises obtaining the predicted fetch time for each portlet for which a thread was spawned in Block 440, and in preferred embodiments, adding a configurable constant value "C" to the largest predicted fetch time, thereby yielding the refresh interval. Alternative approaches may be used without deviating from the scope of the present invention, including but not limited to: adding a fixed (i.e. non-configurable) time to the largest predicted fetch time; using a median fetch time instead of the largest fetch time; and so forth. The computed refresh interval is used to create the refresh header. (Refer to the discussion of Figs.

2A and 2B, above, for more information about the refresh header.)

At Block 430, the portal page is formatted (e.g. as a markup language document), using the content from each portlet that has content currently available (i.e. those portlets in the "ready" state). Preferably, placeholders are inserted for those portlets which are still acquiring their content, as illustrated in Fig. 3A. The formatted document, along with the refresh header when Block 450 was executed, are used to create an HTTP response, which is then returned to the requester.

Note that while references herein are in terms of using HTTP for transmitting request and response messages, this is merely for purposes of illustration. The present invention may be used with other protocols which provide features analogous to those described herein, without deviating from the scope of the present invention. Furthermore, while the present invention is discussed in terms of documents encoded in HTML, this is for purposes of illustration and not of limitation. Other markup languages may be used alternatively, including, but not limited to: Wireless Markup Language ("WML"); i-mode format; and Handheld Device Markup Language ("HDML").

If a subsequent request for this portal page is received in response to the browser's processing of the refresh header, the logic of Fig. 4 is re-executed. If the refresh header was set to a sufficiently large value for the portlets to complete, then the test in Block 420 will have a positive result, and the complete portal page will be formatted and returned in Block 430;

otherwise, a new refresh interval will be computed, and another partially-complete portal page will be sent (in Blocks 450 and 430, respectively). (Note that the processing of Block 440 will be bypassed for those portlets which are in the "loading" state due to an earlier invocation of this logic.)

The flowchart in Fig. 5 illustrates processing occurring on the client. At Block 500, the browser receives a request from the end user to render a portal page. This request may be explicitly entered by the user, or the user might be using this portal page as his or her home page, in which case the request is automatically generated for the user. A request for this portal page (or for individual frames of the portal page, when using a frame-based portal page) is then sent to the portal server (Block 510). After the server's response is received (Block 520), the client renders the returned content (Block 530).

Block 540 checks to see if the response message contained a refresh header. If so, then Block 550 implements a wait for the amount of time specified as a parameter of that header, after which control returns to Block 510 to send another request for the same page/frame. If not, then Block 560 checks to see if revised content has been received from the server using the multipart MIME technique. If so, then the page/frame is repainted (Block 570). In either case, the processing of the current response message is complete (for purposes of the present discussion), and at Block 580, the client waits for the next event.

It will be obvious to one of ordinary skill in the art that the processing depicted in Fig. 5 is simplified for purposes of illustrating the present invention. This processing uses capabilities of prior art browsers, as discussed earlier, thereby facilitating use of the present invention.

Fig. 6 depicts logic operating on a portal server when using the multipart MIME type of the second preferred embodiment. A content request for a portal page (alternatively, for a frame within a portal page, when using frames) is received from a user agent (Block 600). The portlets comprising that portal page/frame are then determined (Block 610), as discussed with reference to Block 410 of Fig. 4. The multipart MIME type header is formatted (Block 620). Block 630 then tests to see if all portlets have their content available (i.e. are in the "ready" state). If so, then a terminating boundary string is to be included in this outbound message, as noted by Block 650. Block 660 sends a response message to the user agent which includes the content currently available (which in this case is the complete portal page/frame), followed by the terminating boundary string. The processing of this request is then complete.

When Block 630 has a negative result, Block 640 formats a response which includes the currently-available content for this portal page/frame, preceded by a boundary string. This response is sent to the requester (Block 670), and separate threads are spawned (Block 680) for each portlet which requires remote content and which is in the "not ready" state. Upon detecting completion of one of these threads, Block 690 checks to see if all portlets are now finished and in the "ready" state. If not, the process of sending another data block which contains the currently-available content preceded by a boundary string is repeated in Blocks 700 and 710, after which the portlet invoker waits for another thread to finish. If all threads have finished, however, then control transfers to Block 650 where a terminating boundary string is included, and the complete version of the portal page/frame is sent to the requester in Block 660.

As has been demonstrated, the present invention provides advantageous techniques for rendering portal pages for users. By delivering a partially-complete page as an immediate response to the page request, and providing programmatically-generated mechanisms for supplying additional content incrementally, the user's experience is improved as contrasted to prior art approaches where the user was forced to wait until all portlet content was ready before seeing anything. When using the present invention, late-finishing portlets can no longer cause the portal page to be delayed in this manner.

As will be appreciated by one of skill in the art, embodiments of the present invention may be provided as methods, systems, or computer program products. Accordingly, the present invention may take the form of an entirely hardware embodiment, an entirely software embodiment or an embodiment combining software and hardware aspects. Furthermore, the present invention may take the form of a computer program product which is embodied on one or more computer-usable storage media (including, but not limited to, disk storage, CD-ROM, optical storage, and so forth) having computer-usable program code embodied therein.

The present invention has been described with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products according to embodiments of the invention. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, special purpose computer, embedded processor or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer-readable memory that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer-readable memory produce an article of manufacture including instruction means which implement the function specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions which execute on the computer or other programmable apparatus provide steps for implementing the functions specified in the flowchart and/or block diagram block or blocks.

While the preferred embodiments of the present invention have been described, additional variations and modifications in those embodiments may occur to those skilled in the art once they learn of the basic inventive concepts. Therefore, it is intended that the appended claims shall be construed to include both the preferred embodiment and all such variations and modifications as fall within the spirit and scope of the invention.